Context-Aware Data Design, Integration, Customization and Tailoring

Context-ADDICT

A Designer Tool for SDR Networks

Fabio Alloni, Giovanni Biscaro

Abstract

This report is an user manual completed by a programmer manual of the Graphical Environment to paint SDR Networks

Keywords: SDR Network ,XML, Relational, Ontology, XML Schema

Released: December 15, 2007
Num. Pages: 20
Internal No.: 2007.06

Context-ADDICT is a project on Context-Aware Data Design, Integration, Customization and Tailoring developed at Politecnico di Milano, Dipartimento di Elettronica e Informazione.
An electronic copy of this document is available for download at:
`http://poseidon.elet.polimi.it/ca/docs/`

A Designer Tool for SDR Networks

Fabio Alloni, Giovanni Biscaro

December 15, 2007

1 Introduction

Our work is a graphical environment that allows to the user to paint SDR networks. This report is an user manual completed by a programmer manual of this program. In the user manual we will describe in details how this program work and all the functionalities supplied to the user. In the programmer manual we will give to the programmer a lot of information about code, and program structure. We will also provide information to apply changes starting from the program arriving to code.

The goal of our work is to provide an user-friendly environment that allows a simple interaction between user and computer.

2 User-Manual

2.1 The network

Networks managed with this tool are directed planar graphs. Loops are allowed and isolated nodes (without any arc) can exist. A graph (or network) is a set of arcs (or edges) and nodes (or vertex). We call arc a directed link between two nodes. There is no distinction between an arc and a directed edge because the link is only directed: we will often use indistinctly these two terms. Each node has a default name composed by 'V' concatenated with a number, each arc has a default name composed by 'E' concatenated with a number, the source node id and the target node id; numbers of nodes or arcs are consecutive. Graphs have a name that is reported on the title of the windows (internal frame). It has a default value: 'New Network' concatenated with a number, but it is possible to change it when the user save the network. Below users will find the description of all the graph's elements. Each element is described by some attributes.

2.1.1 Node properties

The node has three properties:

- **Id**: It is an unique string that identifies every node; it has a 'V' concatenated with a number. This value cannot be changed.
- **Label**: It is the label for the visualization; the default value is the same as the Node-Id, but it is possible to change it.
- **Type**: This field contains the node's logical type; its values can be: 'String, Char, Float, Integer, Double, Real, Int, Boolean, DateTime, PCDATA, ID, ENUM, CDATA'.

2.1.2 Arc properties

Arcs have six properties:

- **Id**:It is a unique string that identifies every arc; it has an 'E' concatenated with a number, the source node id and the target node id. For instance the first arc between 'V1' and 'V2' is like 'E1(V1,V2)'. This value cannot be changed.
- **Label**: It is the label for the visualization; the default value is the same as the Arc-Id but it is possible to change it.
- **distance**: belongs to the real interval [0,1], it is used to indicate how much the concept expressed by T is semantically close to the concept represented by S; in other words, it expresses the capability of the concept associated with T to characterize the concept associated with S
- **relevance**: belongs to the real interval [0,1], it represents the fraction of instances of the concept denoted by S whose complete definition requires at least one instance of the concept denoted by T
- **isKey**: belongs to the integer interval [0,1], is used to indicate that the node T is part of the primary key of S. If the value is 1 the end node will become black.
- **isA**: boolean [0,1], used to indicate if the relation between the two concepts is of the isA kind, expressing a hierarchy generalization.

2.2 Designer Tool

When the application starts, designer-tool (reproduced with the screen-shot) is divided in 4 sections:

1. Menu-Bar with File and Help item;
2. File-Toolbar with standard file management buttons;
3. Some spaces for the networks that we will call Space-Network;
4. Table for element properties that we will call Property-Table;

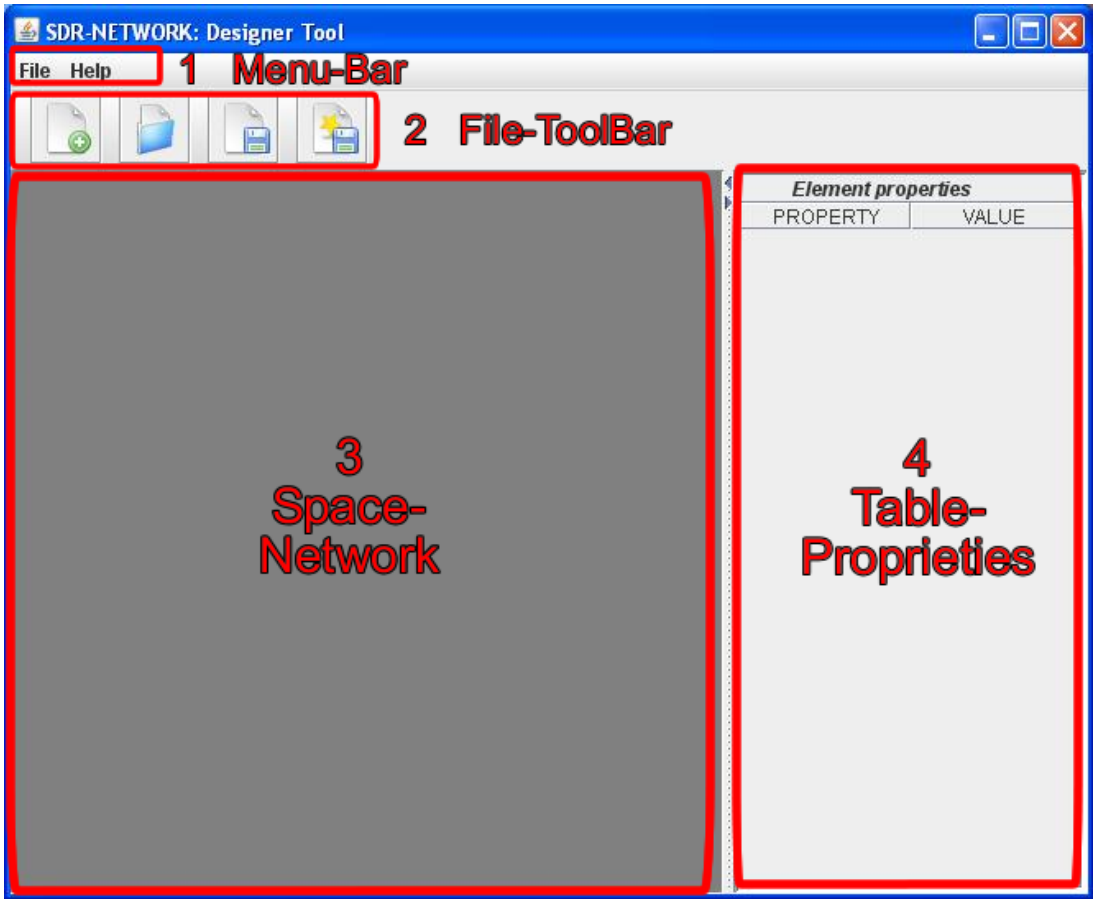


Figura 1: four sections

Menu-Bar and File-Toolbar have the same functionality: they Create, Open, Save and Save As with another name, the file that represents the network.

Space-Network and Property-Table are divided by a split panel. By dragging the divider that appears between the components, the user can specify how much of the split pane's total area goes to each component. The user can also hide one of them with the arrow at the top of the divider. The Property-Table is a table that contains all the properties of the selected element. If users click in some field on, they can change the value of an attribute.

When a file is created or opened, the application changes in an important aspect. At the right side of the File-Toolbar appears the fifth section: a new bar that we will call Network-Toolbar.

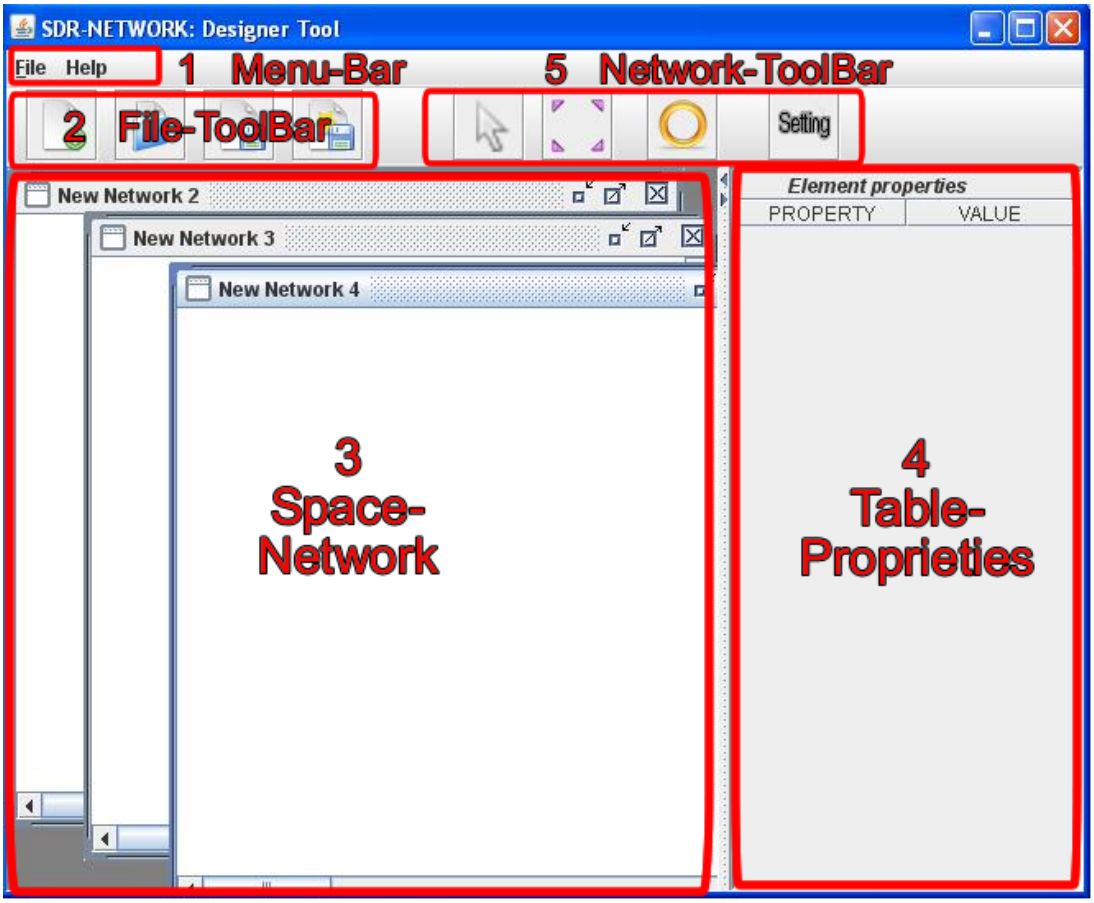


Figura 2: Network-Toolbar

Network-Toolbar contains four buttons for create, select and set the network but it will be explained below.

2.3 Menu-Bar, File-Toolbar and Network-Toolbar

With the Menu-Bar, it is possible to save a network, load a network or create a new one. The descriptions of the Menu-File's items are:

New SDR Network	Create a new window (internal frame) when users create a new network.
Open SDR Network	Load a graph. Show the list of graphs in the default directory. Users have to choose one.
Save SDR Network	Save a graph. If it is not yet saved Users have to choose the name and the directory.
Save As	Save the loaded graph with a new name in a selected directory.

On the one hand the File-Toolbar is a replication of Menu-Bar, in fact it has four buttons and the description is the same as the Menu-Bar; on the other hand the Network-Toolbar has four buttons with four new functionalities which are shown in a table.

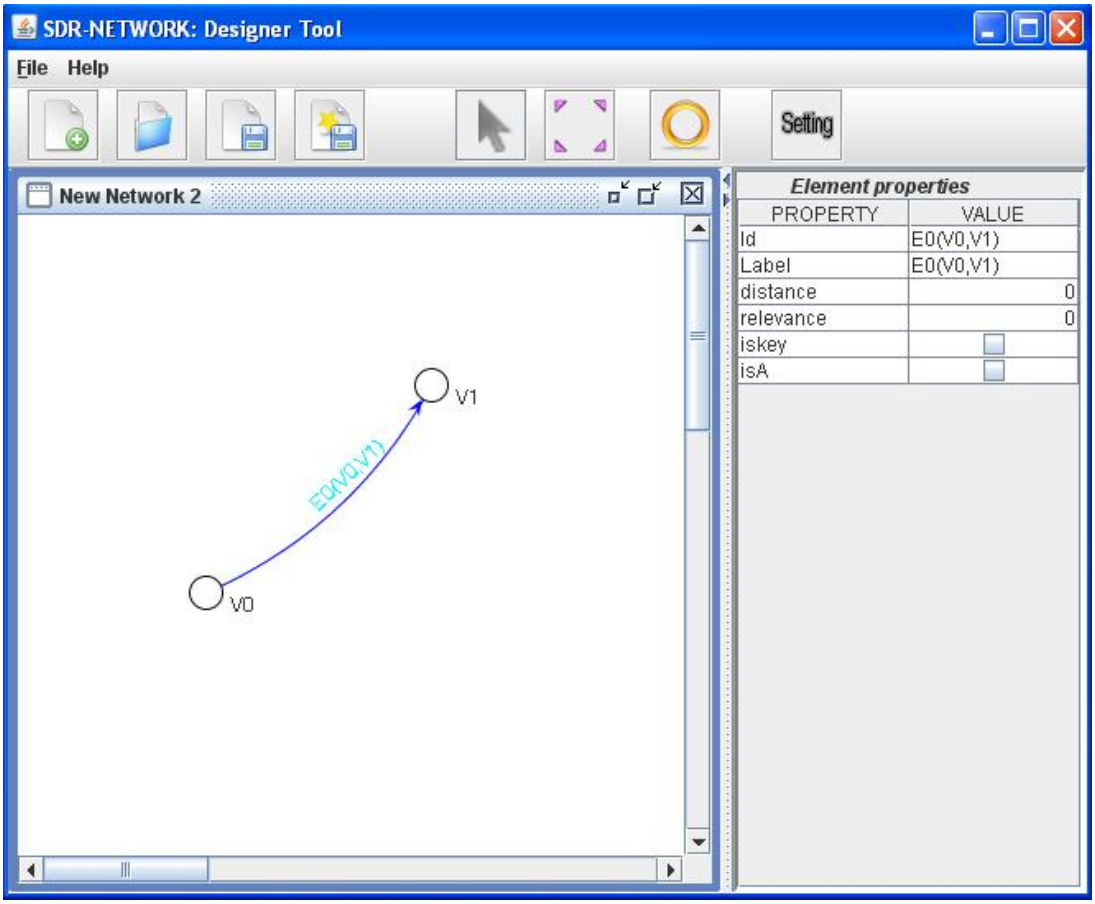
	Selection	enter in a Picking-Mode to manage network's elements
	Panning	enter in a Transforming-Mode to change network's view
	Creating	enter in an Editing-Mode to modify the network
	Settings	change some settings and visual preferences

The 'creating button' operates in two modes: if users push the button, with only one click, after the creation of the new node it becomes unselected; but if users push it with double click, the button becomes selected as long as users click on one other button; In this way users can easily create many nodes without clicking always on 'creating button'.

This Network-Toolbar is logically different for each network that users have opened, but it has the same functionality and property. This choice permits to change settings and its state (editing, selection, transforming...) independently by every open networks. For example the program 'remembers' different visualizations of the arc-label or node-label if users have changed them.

2.4 Managing networks

When a graph is created or loaded, users can choose one of the three modes of the designer-tool; that is for the button (Selection, Panning, and Creating) users have pressed.



2.4.1 Editing-Mode

When users want to create or to modify a network, they select the Editing-Mode. In this way if they press in the white area they create a new node in this position. To create an arc the user should to click on a start node, drag until he is on the end node and then release. If the user releases on a white area nothing happens. Please attention when he wants to create the node, because if he clicks too much close to an other node, the new one will not be created on the other one, but it will be created at security distance. The summary of the typical action in Editing-Mode is:

- MouseButtonOne press on empty space creates a new node
- MouseButtonOne press on a node, followed by a drag to another node creates directed arc between them

2.4.2 Selection-Mode

When users want to move, modify, select elements of the opened network they have to choose the Selection-Mode. In this mode users can select a node or some nodes with 'Shift' and they can move the selection around the empty space. If users want to change the label of nodes or arc they can simply double click on it.

The summary of the typical action in Picking-Mode is:

- MouseButtonOne press on a node or arc to select it
- MouseButtonOne+Shift press on a node or arc to add or toggle selection
- MouseButtonOne+drag on a node to move all selected vertices
- MouseButtonOne+drag on empty space to select node in a rectangle
- MouseButtonOne+Shift+drag on empty space to add to selection with Vertices in a rectangle
- MouseButtonOne double-click on a node or arc to change the label
- MouseButtonOne+ Ctrl press on a node centre it in the window

2.4.3 Transforming-Mode

This mode is only to change the visualization of the network. There are functionalities to a better view of the network, like pan or rotate. The summary of the typical action in Transforming-Mode is:

- MouseButtonOne+drag to translate the display
- MouseButtonOne+Shift+drag to rotate the display
- MouseButtonOne+ctrl(or Command)+drag to shear the display

2.4.4 All modes

There are some operations that users can do in every mode. Each element (node or arc) also has a context menu (mouse menu). Many of the settings on the context menu are duplicated in the Network-Toolbar, but the node or arc deletion is possible only with this solution. Specifically users can delete the single or multiple arcs with 'Delete Edge/Edges' or they can delete the single or multiple nodes with 'Delete Node/Nodes', but this last action cause the deletion of all the incident arc. To create a new node users can click in the white area of the network with a right-button of the mouse, so appear the mouse menu with 'Create Node'. The summary of the action in all modes is:

- MouseButtonThree press on an empty space shows a popup menu to create a Node
- MouseButtonThree press on a Node shows a popup to delete the node

  If any nodes are picked, the popup menu allow creation of an arc.
- MouseWheel to scale. When scale value < 1, view is scaled. When scale value > 1, layout is scaled.

2.5 Settings

When users click on the button Settings on Network-Toolbar appears a small window with three tags:

2.5.1 Node-Tag

In this case users can change the visualization setting of all the nodes:

- **Display the name of the node**: If this is checked, in the window next to the node, appears a label with the name of the node.
- **Display the type of the node**: If this is checked, in the window next to the node, appears a label with the type of the node.

If all of the checks are clicked (name and type), they appear concatenated by underscore

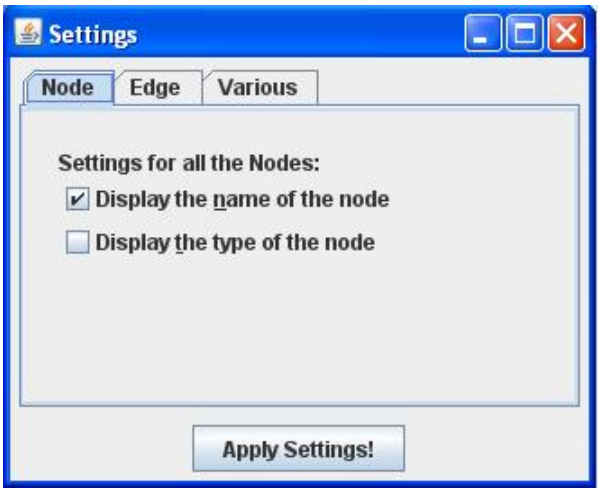


Figura 3: Node Tag

2.5.2 Edge-Tag

In this case users can change the visualization settings for of all the arcs. There are five properties that can be viewed; the default setting shows only the name of the arc, but if users select Distance, Relevance, isKey or isA attributes this information are concatenated by underscore. Obviously if users do not select anything there is not any arc-label on the screen.

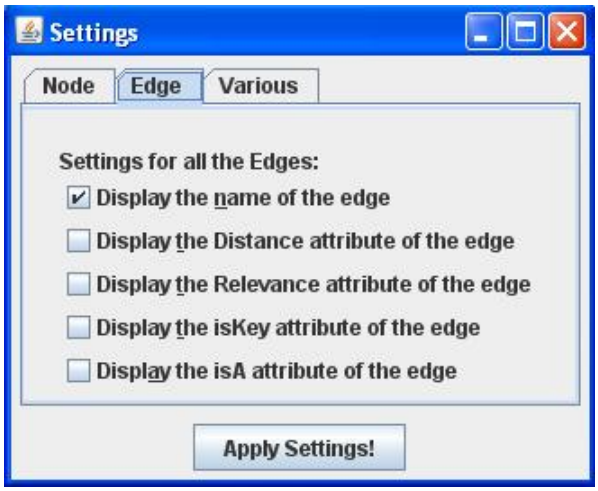

Settings
Node
Edge
Various
Settings for all the Edges:
Display the name of the edge
Display the Distance attribute of the edge
Display the Relevance attribute of the edge
Display the isKey attribute of the edge
Display the isA attribute of the edge
Apply Settings!

Figura 4: Edge Tag

2.5.3 Various-Tag

In this tag there are only two visualization options of the network:

- **Label position**: This option defines the position (in percent along a slide bar) of all labels of the network, compared to the entire width of the arc. For example if the slider's position is completely at right, labels are all near the point of the arrow.
- **Edge Form**: This option defines the form of the edge. For example if the cursor of the slider is completely at right position, arcs have the max spaciousness, but if the cursor is completely at left position, the arc is straight.

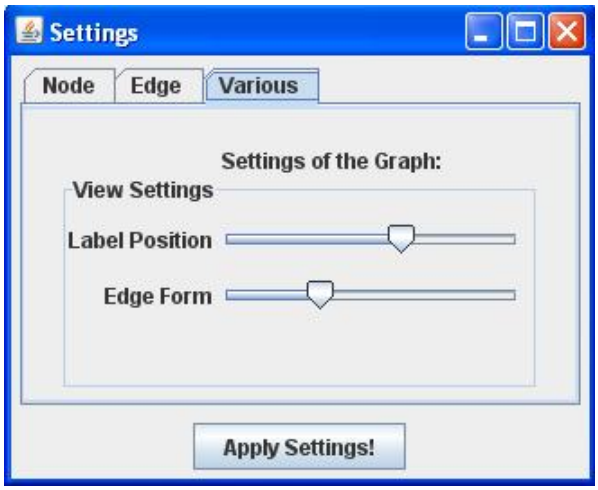


Figura 5: Various Tag

3 Programmer's manual

First of all, this section contains a description of the Jung library. This library is the most important one used into program. Also, we will describe the structural organization of the software. For each java class, a short description is also provided. Then, we will explain some tricks to apply changes to the software, starting from from the designer arriving to the code.

3.1 The Jung Library

JUNG - the Java Universal Network/Graph Framework - is a software library that provides a common and extensible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. It is written in Java, which allows JUNG-based applications to make use of the extensive built-in capabilities of the Java API, as well as those of other existing third-party Java libraries.

The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, multi-modal graphs, graphs with parallel edges, and hypergraphs. It provides a mechanism for annotating graphs, entities, and relations with metadata. This facilitates the creation of analytic tools for complex data sets that can examine the relations between entities as well as the metadata attached to each entity and relation.

The current distribution of JUNG includes implementations of a number of algorithms from graph theory, data mining, and social network analysis, such as routines for clustering, decomposition, optimization, random graph generation, statistical analysis, and calculation of network distances, flows, and importance measures (centrality, PageRank, HITS, etc.).

JUNG also provides a visualization framework that makes it easy to construct tools for the interactive exploration of network data. Users can use one of the layout algorithms provided, or use the framework to create their own custom layouts. In addition, filtering mechanisms are provided which allow users to focus their attention, or their algorithms, on specific portions of the graph.

As an open-source library, JUNG provides a common framework for graph/network analysis and visualization. JUNG makes it easier for those who work with relational data to make use of one anothers' development efforts, and thus avoid continually re-inventing the wheel.

3.2 The system architecture

Our network designer is a java written application. So, the software is organized into packages. Our application is divided into two packages. First, named "gui", contains all classes needed for running the graphical user interface. Second package, named "structure" contains two classes that are responsible for managing the logical model of our network (class SDRNetwork) and of the entire application (SDRManager).

3.2.1 GUI package

This package is responsible for all the operations that concern visualization and user interaction experience. Now we provide a short description of each class contained in this package.

- `About.java`: this is a simple class that provide general information of the designer (such as author and version) to the user.
- `FloatEditor.java`: this class is responsible for the correct insertion of number into application. It provides some error message to the user when the insertion fail, and it manages further action.
- `MainFrame.java`: this contains the code to paint the main window on the screen and it initializes other structure such as logs, bars and panels. This class contains the main method that allows to start the program.
- `MenuBar.java`: this file provides the menu bar that allows to the user to perform some standard operation such as (create new network, open and save file, exit the program, etc...)
- `MyEdgePaint.java`: it performs the choice of which color have to use to render the edges. Normally the color is black, but the edge can be rendered with different color according to the user action.
- `MyEditingMousePlugin.java`: Jung performs mouse control in a very interesting way. It is provide a simple bottom layer on which the user can insert some plugins to execute different operations. Editing mouse plugin is a class that perform the insertion on the canvas of nodes and edges. Not only, it also allows to the user to modify edge or node label, when a double click action is made. This class reacts to the input event that are fired when the user uses the mouse.
- `MyEditingPopupGraphMousePlugin.java`: this plugin performs the operation that can be done when the user clicks the right button of his mouse. Few action can be execute. It is possible to insert node or edge, or to delete it.
- `MyGraphMouse.java`: this class represents the bottom layer for managing mouse control we have discussed before. It extends the general purpose class provided by the Jung library. In this way, the programmer can insert his personal feature into program.
- `MyLayout.java`: it is responsible for the spatial location of nodes on the canvas. This class exploits the functionalities offered by the spring layout implementation provided by Jung. This implementation confers a dynamical behavior to the nodes on canvas.
- `MyLocationDecorator.java`: this class performs some mathematical computation to randomize the position of a node on canvas, when the position in not provided otherwise.
- `MyPaintFunction.java`: Similar to MyEdgePaint.java, this class select the color to render the nodes. The color is not only a graphical decoration. In a SDR network, the color of node has also an important semantic value.
- `MyPickingMousePlugin.java`: this plugin reacts to the selection actions performed by the user. It allows multiple selection in order to execute collective operations. It is also responsible to manage dragging operation.
- `PropertyModel.java`: This class contains the table's model that allows to the user to insert values for the parameters. The table's model collects the data inserted by the user o read from memory.

- `PropertyTable.java`: This file contains the code necessary to render the table. Boolean values are rendered as a checkbox, numerical values are rendered with the suitable rendered described before, string values are rendered with text field.
- `PropertyPanel.java`: Every graphical object needs a panel to be rendered. This is the panel to render the table. This class reacts to changes on table and it performs the right operation.
- `SDRBar.java`: this bar contains some buttons that allow to the user, to interact with the environment. This bar contains only the buttons that are related to the action can be performed on a single network. Every network has its bar.
- `SDRDesigner.java`: this class contains the main panel of the graphical user interface, and it represents the root of the application. In this file, bars and information panels are also initialized. This panel can be plugged into other application because it extends the JComponent class.
- `SDRInternalFrame.java`: this piece of code contains the internal window that allows to user to paint graph. For this reason, this class can be correctly considered the most important class of the entire visualization system. Here, are initialized a lot of component that are useful to work.
- `SettingsFrame.java`: this class implements a small window that allows to user to modify some parameters. Composition of nodes and edges labels are adjustable. It is also modifiable the label position and the shape of the edges.
- `StandardBar.java`: this bar contains buttons that are useful for creating a new empty sheet, opening an existing network and saving the created one. This bar is always visible and accessible.

3.2.2 Structure package

This package is composed by only two classes, responsible for managing and storing information about networks.

- `SDRManager.java`: this class contains the skeleton of the application. It contains the list of all the networks are opened and include methods to create new networks or to close an existing one. The current state of the environment is always correctly maintained by this class. Code for loading networks from files is contained here. The summary of fields and methods contained is displayed in the figure below.

Figura 6: SDRManager class diagram

- `SDRNetwork.java`: this piece of code contains the graph model used by the application. This graph model, provided by Jung library, is called "Sparse Graph". It allows to create cyclic and acyclic graphs with directed and undirected edges. Multiple edges between two nodes are also permitted. To create our application we have to restrict this model by removing undirected edges that are forbidden in SDR networks. This implementation stores nodes and edges into sets and it becomes much efficient when graphs are sparse. This class also provides methods to initialize and to store graphs. The summary of fields and methods is displayed in the picture below.

Figura 7: SDRNetwork class diagram

3.3 Reverse manual

In this section we present to programmer, how apply some simple changes to the code, starting from the environment.

The changes we will describe are:

- Changing color to a node
- Changing admissible values to the "type" parameter of a node
- Adding a new parameter to a node or to an edge

Changing color to a node

In a SDR network, the colors of nodes are not only graphical decorations, but they have an important semantic value. It is reasonable to think that colors in the future may be updated. For this reason we describe in this section how to change color.

In a SDR network a node can assume four different color

- white: normal node
- black: a node becomes black when an incoming edge have "isKey" value set to true.
- yellow: normal node selected
- orange: black node selected

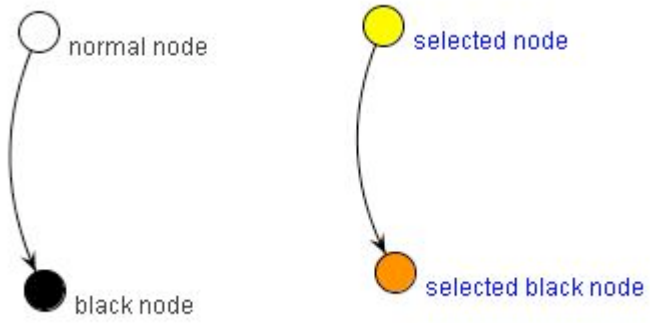


Figura 8: Node colors

The class that is responsible for assigning colors to nodes is the MyPaintFunction.java. This class contains the getFillPaint method which is called one times for each node every canvas refresh. The code for this method is reported in algorithm 1. The programmer can easily modify colors or conditions that determine colors changing this code.

Algorithm 1 Get fill paint method

```
    public Paint getFillPaint(Vertex v) {
    boolean found = false;
    Set set = v.getInEdges();
    Iterator it = set.iterator();
    while (it.hasNext()) {
       Edge edge = (Edge) it.next();
       boolean isKey = false;
       try {
          isKey = ((Boolean) edge.getUserDatum("isKey"))
             .booleanValue();
       } catch (Exception e1) {
          isKey = (new Boolean((String)edge
             .getUserDatum("isKey"))).booleanValue();
          edge.setUserDatum("isKey", isKey,
             new UserDataContainer.CopyAction.Clone());
       }
       if (isKey) {
          found = true;
          break;
       }
    }
    if (pick.isPicked(v) && found) {
       return new Color(255,128,0);
    } if (pick.isPicked(v) && !found) {
       return Color.YELLOW;
    } if (found) {
       return Color.BLACK;
    } else
       return Color.WHITE;
}
```

Changing admissible values to the "type" parameter of a node

Normally, to change admissible values to a parameter it is necessary to directly modify the code. In this case, instead, all the values are extracted from the code and they are put into an external `xml` file. To change the values the programmer have to modify only the `config/settings.xml` file whose content is reported below

Algorithm 2 settings.xml file

```
<settings>
  <data_type>
      <type>none</type>
      <type>String</type>
      <type>Char</type>
      <type>Float</type>
      <type>Integer</type>
      <type>Double</type>
      <type>Real</type>
      <type>Int</type>
      <type>Boolean</type>
      <type>Datetime</type>
      <type>PCDATA</type>
      <type>ID</type>
      <type>ENUM</type>
      <type>CDATA</type>
  </data_type>
</settings>
```

Adding a new parameter to a node or to an edge

To collect the parameters of an element, the Jung library uses the user data class. It represents custom user and system level information to extend the definition of a node, an edge or a graph. This is the easiest way to extend the class without subclassing. So, the programmer can easily add a new parameter (or modify the value of an existing one) calling the "setUserDatum" method. This method needs three parameter: the key or the name of the parameter, the parameter itself, and the action to do during copy action. In order to recover the value of the parameter the programmer has to use the "getUserDatum" method. It needs only one parameter: the key. In this way, the programmer has nothing to do to save or load the new parameter. Our system loads and saves all the parameters that are contained in the user data container.

Riferimenti bibliografici

[1] http://poseidon.elet.polimi.it/ca/
Context-Aware Data Design, Integration, Customization, Tailoring and Management for Very Small, Portable Databases

[2] http://java.sun.com/
The source for Java Developers

[3] http://jung.sourceforge.net/
Java Universal Network/Graph Framework

[4] http://www.w3.org/XML/
Extensible Markup Language (XML)

[5] http://java.sun.com/docs/books/tutorial/
The Java Tutorials

[6] http://www.eclipse.org/
Eclipse - an open development platform

[7] http://graphml.graphdrawing.org/
GraphML, a comprehensive and easy-to-use file format for graphs.